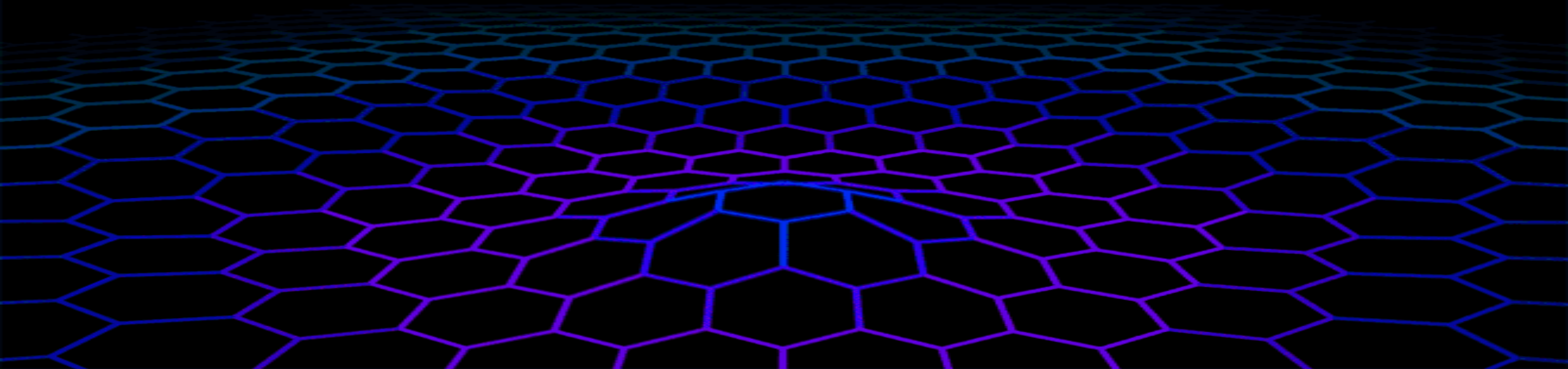


Banco de Dados 1



JOINS

Utilizando JOINS

Utilizar a cláusula WHERE para fazer seus JOINS (relacionamentos), limita os relacionamentos a apenas um tipo deles, o INNER JOIN.

Temos três tipos de Joins:

INNER JOIN

LEFT JOIN

RIGHT JOIN

INNER JOIN

Retorna apenas as linhas das tabelas que sejam comuns entre si, ou seja, as linhas em ambas as tabelas que possuam o campo de relacionamento com o mesmo valor.

LEFT JOIN

Irá listar todas as linhas da primeira tabela relacionada no JOIN, logo após a cláusula FROM.

Quando a linha listada não possuir equivalência na tabela destino , as colunas da tabela destino aparecerão com valores nulos

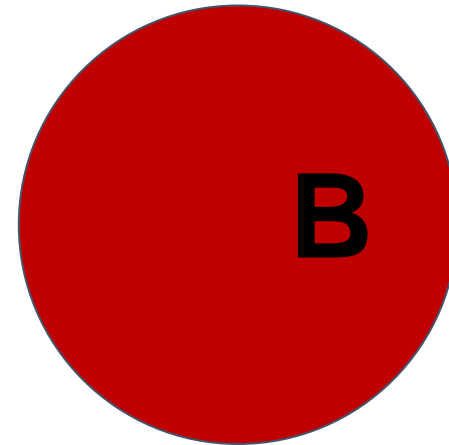
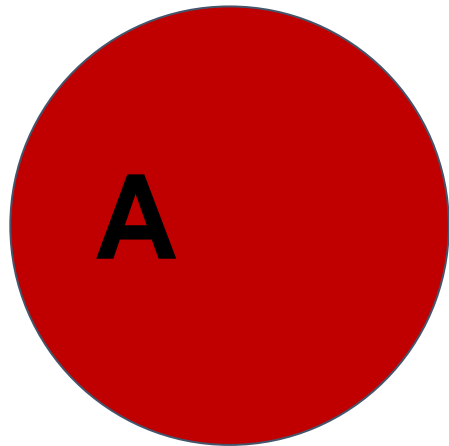
RIGHT JOIN

Irá listar todas as linhas referentes à segunda tabela relacionada no JOIN

Neste caso também , quando a linha listada não possuir equivalência na tabela destino , as colunas da tabela destino aparecerão com valores nulos

Comandos SQL – DML - JOIN

Suponha que você tem duas tabelas, **A** e **B**



Comandos SQL – DML - JOIN

Os dados destas tabelas são exibidos a seguir:

A

id	valor
1	FOX
2	COP
3	TAXI
6	WASHINGTON
7	DELL
5	ARIZONA
4	LINCOLN
10	LUCENT

8 rows in set (0.00 sec)

B

id	valor
1	TROT
2	CAR
3	CAB
6	MONUMENT
7	PC
8	MICROSOFT
9	APPEL
11	SCOTCH

8 rows in set (0.00 sec)

Tabelas A e B - criação

```
DROP DATABASE IF EXISTS aula13;  
CREATE DATABASE IF NOT EXISTS aula13;  
USE aula13;
```

```
CREATE TABLE a (  
    id      INT(11),  
    valor   CHAR(30)  
);
```

```
CREATE TABLE b (  
    id      INT(11),  
    valor   CHAR(30)  
);
```

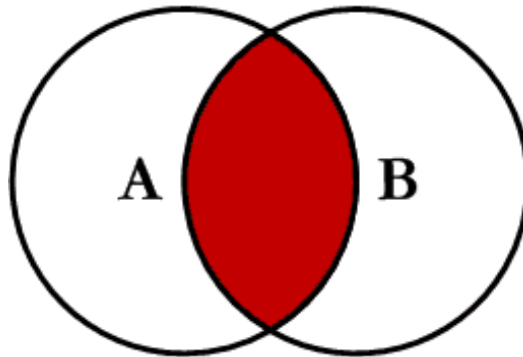
Tabelas A e B - conteúdo

```
INSERT INTO a VALUES(1, "FOX");
INSERT INTO a VALUES(2, "COP");
INSERT INTO a VALUES(3, "TAXI");
INSERT INTO a VALUES(6, "WASHINGTON");
INSERT INTO a VALUES(7, "DELL");
INSERT INTO a VALUES(5, "ARIZONA");
INSERT INTO a VALUES(4, "LINCOLN");
INSERT INTO a VALUES(10, "LUCENT");
```

```
INSERT INTO b VALUES(1, "TROT");
INSERT INTO b VALUES(2, "CAR");
INSERT INTO b VALUES(3, "CAB");
INSERT INTO b VALUES(6, "MONUMENT");
INSERT INTO b VALUES(7, "PC");
INSERT INTO b VALUES(8, "MICROSOFT");
INSERT INTO b VALUES(9, "APPEL");
INSERT INTO b VALUES(11, "SCOTCH");
```

INNER JOIN

Retorna todos os registros na tabela à esquerda (tabela A) que tem um registro correspondente na tabela direita (tabela B).



INNER JOIN

```
SELECT *  
FROM a  
INNER JOIN b ON a.id = b.id;
```

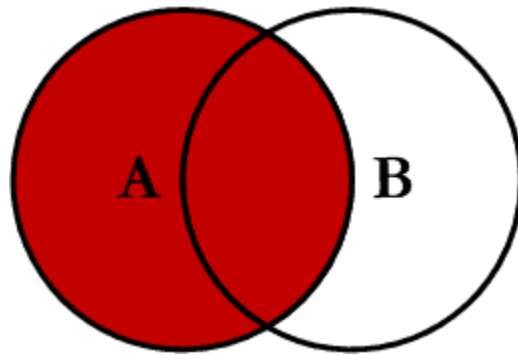
```
SELECT *  
FROM a,  
      b  
WHERE a.id = b.id;
```

INNER JOIN

a.id	a.valor	b.valor	b.id
1	FOX	TROT	1
2	COP	CAR	2
3	TAXI	CAB	3
6	WASHINGTON	MONUMENT	6
7	DELL	PC	7

LEFT JOIN

Retorna todos os registros na tabela esquerda (tabela A), independentemente se algum desses registros têm uma correspondência na tabela direita (tabela B). Também retornará os registros correspondentes na tabela direita.



LEFT JOIN

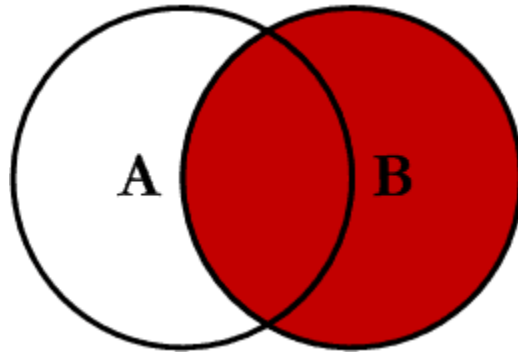
```
SELECT *  
FROM a  
LEFT JOIN b ON a.id = b.id;
```

LEFT JOIN

a.id	a.valor	b.valor	b.id
1	FOX	TROT	1
2	COP	CAR	2
3	TAXI	CAB	3
6	WASHINGTON	MONUMENT	6
7	DELL	PC	7
5	ARIZONA	NULL	NULL
4	LINCOLN	NULL	NULL
10	LUCENT	NULL	NULL

RIGHT JOIN

Retorna todos os registros na tabela direita (tabela B), independentemente se algum desses registros têm uma correspondência na tabela da esquerda (tabela A). Também retornará os registros correspondentes da tabela à esquerda.



RIGHT JOIN

```
SELECT *  
FROM a  
RIGHT JOIN b ON a.id = b.id;
```

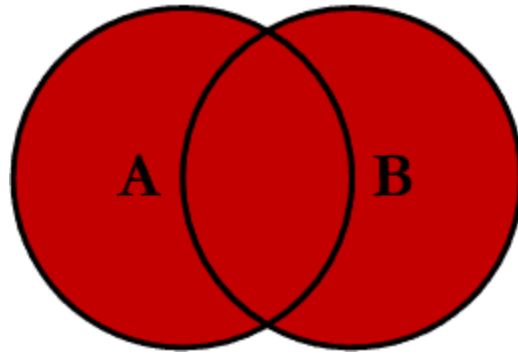
RIGHT JOIN

a.id	a.valor	b.valor	b.id
1	FOX	TROT	1
2	COP	CAR	2
3	TAXI	CAB	3
6	WASHINGTON	MONUMENT	6
7	DELL	PC	7
NULL	NULL	MICROSOFT	8
NULL	NULL	APPLE	9
NULL	NULL	SCOTCH	11

OUTER JOIN

Este JOIN também pode ser chamado de FULL OUTER JOIN ou FULL JOIN.

Retorna todos os registros de ambas as tabelas, juntando-se os registros da tabela da esquerda (tabela A) que os registros correspondentes na tabela da direita (tabela B).



OUTER JOIN

Obs.: O MySQL não possui o comando OUTER JOIN.

Ele pode ser implementado através de um UNION entre um LEFT e um RIGHT JOIN.

OUTER JOIN

Exemplo em SQL

```
SELECT *  
FROM a  
FULL OUTER JOIN b  
ON a.id = b.id;
```

OUTER JOIN

Exemplo (compatível) em MySQL

```
SELECT *  
FROM a  
LEFT JOIN b ON a.id = b.id
```

UNION

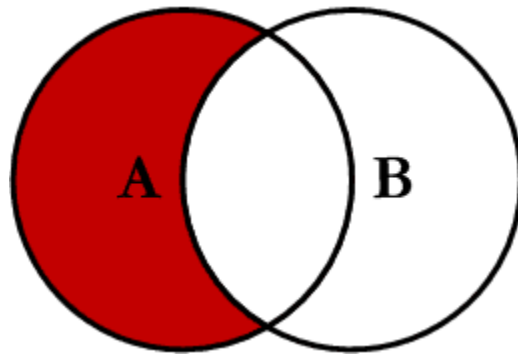
```
SELECT *  
FROM a  
RIGHT JOIN b ON a.id = b.id;
```

OUTER JOIN

a.id	a.valor	b.valor	b.id
1	FOX	TROT	1
2	COP	CAR	2
3	TAXI	CAB	3
6	WASHINGTON	MONUMENT	6
7	DELL	PC	7
5	ARIZONA	NULL	NULL
4	LINCOLN	NULL	NULL
10	LUCENT	NULL	NULL
NULL	NULL	MICROSOFT	8
NULL	NULL	APPLE	9
NULL	NULL	SCOTCH	11

LEFT EXCLUDING JOIN

Retorna todos os registros na tabela à esquerda (tabela A), que não possuam nenhum registro correspondente na tabela direita (tabela B).



LEFT EXCLUDING JOIN

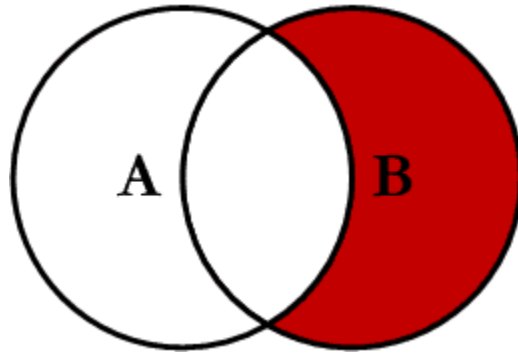
```
SELECT *  
FROM a  
LEFT JOIN b ON a.id = b.id  
WHERE b.id IS NULL;
```

LEFT EXCLUDING JOIN

a.id	a.valor	b.valor	b.id
5	ARIZONA	NULL	NULL
4	LINCOLN	NULL	NULL
10	LUCENT	NULL	NULL

RIGHT EXCLUDING JOIN

Retorna todos os registros na tabela direita (tabela B) que não possuam nenhum registro correspondente na tabela à esquerda (tabela A).



RIGHT EXCLUDING JOIN

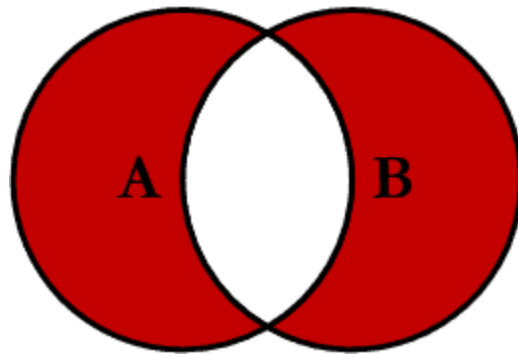
```
SELECT *  
FROM a  
RIGHT JOIN b ON a.id = b.id  
WHERE a.id IS NULL;
```

RIGHT EXCLUDING JOIN

a.id	a.valor	b.valor	b.id
NULL	NULL	MICROSOFT	8
NULL	NULL	APPLE	9
NULL	NULL	SCOTCH	11

OUTER EXCLUDING JOIN

Retorna todos os registros na tabela à esquerda (tabela A) e todos os registros na tabela direita (tabela B) que não correspondem.



OUTER EXCLUDING JOIN

Exemplo em SQL

```
SELECT *  
FROM a  
FULL OUTER JOIN b  
ON a.id = b.id  
WHERE a.id IS NULL  
OR b.id IS NULL;
```


OUTER EXCLUDING JOIN

Exemplo (compatível) em MySQL

```
SELECT *  
FROM a  
LEFT JOIN b ON a.id = b.id  
WHERE b.id IS NULL
```

UNION

```
SELECT *  
FROM a  
RIGHT JOIN b ON a.id = b.id  
WHERE a.id IS NULL;
```

OUTER EXCLUDING JOIN

a.id	a.valor	b.valor	b.id
5	ARIZONA	NULL	NULL
4	LINCOLN	NULL	NULL
10	LUCENT	NULL	NULL
NULL	NULL	MICROSOFT	8
NULL	NULL	APPLE	9
NULL	NULL	SCOTCH	11

Comandos SQL – DML - JOIN

Resumo



Crie as tabelas conforme solicitado

CLI

Codigo - INTEIRO - AUTO NUMERAÇÃO – CHAVE
Nome – CHAR (30)

CLI	
Codigo	Nome
1	José
2	Elisio
3	Roberto
4	Guilherme

PEDIDO

nr - INTEIRO – CHAVE
cliente – INTEIRO
valor – DECIMAL(9,2)

PEDIDO		
nr	Cliente	valor
1	2	100.50
2	2	120.00
3	1	20.00
4	3	60.00
5	3	110.00

Crie as tabelas conforme solicitado

```
CREATE TABLE cli (  
    codigo INT(11) AUTO_INCREMENT,  
    nome CHAR(30),  
    PRIMARY KEY (codigo)  
);
```

```
CREATE TABLE pedido (  
    nr INT(11),  
    cliente INT(11),  
    valor DECIMAL(9,2),  
    PRIMARY KEY (nr)  
);
```

Crie as tabelas conforme solicitado

```
INSERT INTO cli (nome) VALUES("José");  
INSERT INTO cli (nome) VALUES("Elísio");  
INSERT INTO cli (nome) VALUES("Roberto");  
INSERT INTO cli (nome) VALUES("Guilherme");
```

```
INSERT INTO pedido (nr, cliente, valor) VALUES(1, 2, 100.50);  
INSERT INTO pedido (nr, cliente, valor) VALUES(2, 2, 120.00);  
INSERT INTO pedido (nr, cliente, valor) VALUES(3, 1, 20.00);  
INSERT INTO pedido (nr, cliente, valor) VALUES(4, 3, 60.00);  
INSERT INTO pedido (nr, cliente, valor) VALUES(5, 3, 110.00);
```


INNER JOIN

```
SELECT pedido.nr,  
       cli.nome,  
       pedido.valor  
FROM cli  
INNER JOIN pedido ON (cli.codigo = pedido.cliente);
```

Observe que o cliente **Guilherme** não fez nenhum pedido.

nr	nome	valor
1	Elísio	100.50
2	Elísio	120.00
3	José	20.00
4	Roberto	60.00
5	Roberto	110.00

5 rows in set (0.00 sec)

Observe que o cliente Guilherme não fez nenhum pedido.

E se você quiser um relatório que mostre também os clientes que não fizeram nenhum pedido?

Você terá que usar uma junção chamada **LEFT JOIN**:

```
SELECT pedido.nr,  
       cli.nome,  
       pedido.valor  
FROM cli  
LEFT JOIN pedido ON (cli.codigo = pedido.cliente);
```

Veja que agora o cliente que não fez nenhum pedido, no caso o **Guilherme**, também foi exibido.

nr	nome	valor
3	José	20.00
2	Elísio	120.00
1	Elísio	100.50
5	Roberto	110.00
4	Roberto	60.00
NULL	Guilherme	NULL

6 rows in set (0.00 sec)

Se quiser exibir apenas os clientes que não fizeram nenhum pedido, use:

```
SELECT pedido.nr,  
       cli.nome,  
       pedido.valor  
FROM cli  
LEFT JOIN pedido ON (cli.codigo = pedido.cliente)  
WHERE pedido.nr IS NULL;
```

```
+-----+-----+-----+  
| nr    | nome      | valor |  
+-----+-----+-----+  
| NULL  | Guilherme | NULL  |  
+-----+-----+-----+  
1 row in set (0.00 sec)
```

Lance mais alguns Clientes na tabela CLI mas não lance Pedidos para eles.

```
INSERT INTO cli (nome) VALUES("Elaine");  
INSERT INTO cli (nome) VALUES("Maria");  
INSERT INTO cli (nome) VALUES("Thais");
```

```
SELECT pedido.nr,  
       cli.nome,  
       pedido.valor  
FROM cli  
LEFT JOIN pedido ON (cli.codigo = pedido.cliente);
```

nr	nome	valor
3	José	20.00
2	Elísio	120.00
1	Elísio	100.50
5	Roberto	110.00
4	Roberto	60.00
NULL	Guilherme	NULL
NULL	Elaine	NULL
NULL	Maria	NULL
NULL	Thais	NULL

9 rows in set (0.00 sec)

Abaixo vemos exemplos do uso de COUNT () :

```
SELECT COUNT(pedido.valor)
FROM cli
LEFT JOIN pedido ON (cli.codigo = pedido.cliente);
```

```
+-----+
| COUNT(valor) |
+-----+
|           5 |
+-----+
1 row in set (0.00 sec)
```

```
SELECT COUNT(*)
FROM cli
LEFT JOIN pedido ON (cli.codigo = pedido.cliente);
```

```
+-----+
| COUNT(*) |
+-----+
|         9 |
+-----+
1 row in set (0.00 sec)
```

Alias para Tabelas

- Quando usamos JOIN, o nome da tabela é citado para diferenciar a qual campo se está fazendo referência. Quando a consulta é complexa e envolve várias tabelas, referenciar o nome da tabela pode aumentar muito o tamanho da consulta e em algumas ferramentas como Delphi, há um limite de 255 caracteres para a consulta.
- Para criar um *alias* para uma tabela, basta acrescentar um identificador à frente do nome da tabela. A partir de então, basta utilizar este *alias* para se referenciar à tabela.

```
SELECT nome, nr, valor FROM pedido p INNER JOIN cli c ON (p.cliente=c.codigo);
```



Exercícios

Criar a tabela **proprietarios** e a tabela **carros**:

```
CREATE TABLE proprietarios(  
    rg      CHAR(16),  
    nome    VARCHAR(40),  
    PRIMARY KEY(rg)  
);
```

```
CREATE TABLE carros(  
    renavam    CHAR(12),  
    modelo     VARCHAR(20),  
    marca      VARCHAR(20),  
    cor        VARCHAR(10),  
    proprietario_rg CHAR(16),  
    PRIMARY KEY(renavam)  
);
```

Exercícios

1. Insira os seguintes valores para proprietários:

rg	nome
123456789	João da Silva
654123987	Maria de Oliveira
987654321	José de Souza

3 rows in set (0.00 sec)

2. Insira os seguintes valores para carros:

renavam	modelo	marca	cor	proprietario_rg
123456789123	Fiesta	Ford	Prata	123456789
123456789124	Palio	Fiat	Vermelho	123456789
123456789125	Corsa	Chevrolet	Amarelo	987654321
123456789126	Gol	Volkswagen	Branco	987654321

4 rows in set (0.00 sec)

Exercícios

```
-- Insira os seguintes valores para proprietários
INSERT INTO proprietarios VALUES("123456789", "João da Silva");
INSERT INTO proprietarios VALUES("654123987", "Maria de Oliveira");
INSERT INTO proprietarios VALUES("987654321", "José de Souza");

-- Insira os seguintes valores para carros
INSERT INTO carros VALUES("123456789123", "Fiesta", "Ford", "Prata", "123456789");
INSERT INTO carros VALUES("123456789124", "Palio", "Fiat", "Vermelho", "123456789");
INSERT INTO carros VALUES("123456789125", "Corsa", "Chevrolet", "Amarelo", "987654321");
INSERT INTO carros VALUES("123456789126", "Gol", "Volkswagen", "Branco", "987654321");
```

Exercícios

3 - Listar o Renavam, modelo, marca, cor e nome do proprietário de todos os carros.

renavam	modelo	marca	cor	nome
123456789124	Palio	Fiat	Vermelho	João da Silva
123456789123	Fiesta	Ford	Prata	João da Silva
NULL	NULL	NULL	NULL	Maria de Oliveira
123456789126	Gol	Volkswagen	Branco	José de Souza
123456789125	Corsa	Chevrolet	Amarelo	José de Souza

5 rows in set (0.00 sec)

Exercícios

3 - Listar o Renavam, modelo, marca, cor e nome do proprietário de todos os carros.

```
SELECT c.renavam,  
       c.modelo,  
       c.marca,  
       c.cor,  
       p.nome  
FROM proprietarios p  
LEFT JOIN carros c ON p.rg = c.proprietario_rg;
```

renavam	modelo	marca	cor	nome
123456789124	Palio	Fiat	Vermelho	João da Silva
123456789123	Fiesta	Ford	Prata	João da Silva
NULL	NULL	NULL	NULL	Maria de Oliveira
123456789126	Gol	Volkswagen	Branco	José de Souza
123456789125	Corsa	Chevrolet	Amarelo	José de Souza

5 rows in set (0.00 sec)

Exercícios

1 - Exiba quantos carros tem cada proprietário.

2 - Exibir quantos carros tem cada proprietário **que possui carros**, ou seja, quem não possui nenhum carro não deve ser exibido.

Exercícios

1 - Exiba quantos carros tem cada proprietário.

```
SELECT p.nome,  
       COUNT(c.renavam)  
FROM proprietarios p  
LEFT JOIN carros c ON p.rg = c.proprietario_rg  
GROUP BY p.nome;
```

2 - Exibir quantos carros tem cada proprietário **que possui carros**, ou seja, quem não possui nenhum carro não deve ser exibido.

```
SELECT p.nome          AS Proprietário,  
       COUNT(c.renavam) AS qtd_veículos  
FROM proprietarios p  
INNER JOIN carros c ON p.rg = c.proprietario_rg  
GROUP BY p.nome;
```

Voltando agora a trabalhar com as tabelas já feitas anteriormente:

CLI

codigo	nome
1	José
2	Elísio
3	Roberto
4	Guilherme
5	Elaine
6	Maria
7	Thais

7 rows in set (0.00 sec)

PEDIDO

nr	cliente	valor
1	2	100.50
2	2	120.00
3	1	20.00
4	3	60.00
5	3	110.00

5 rows in set (0.00 sec)

Para se certificar de ter essas tabelas execute o comando SHOW TABLES; e depois os comandos SELECT * para cada uma delas.

Execute o comando:

```
SELECT cli.nome,  
       pedido.nr,  
       pedido.valor  
FROM pedido  
INNER JOIN cli ON pedido.cliente = cli.codigo;
```

nome	nr	valor
Elísio	1	100.50
Elísio	2	120.00
José	3	20.00
Roberto	4	60.00
Roberto	5	110.00

5 rows in set (0.00 sec)

Exercício

Exibir a quantidade e o valor total dos pedidos por cliente:

Exercício

Exibir a quantidade e o valor total dos pedidos por cliente:

```
SELECT c.nome          AS Cliente,  
       COUNT(p.valor)  AS Quantidade,  
       SUM(p.valor)    AS Total  
FROM cli c  
LEFT JOIN pedido p ON c.codigo = p.cliente  
GROUP BY c.nome  
ORDER BY c.nome;
```

Cliente	Quantidade	Total
Elaine	0	NULL
Elisio	2	220.50
Guilherme	0	NULL
Jose	1	20.00
Maria	0	NULL
Roberto	2	170.00
Thais	0	NULL

7 rows in set (0.00 sec)

Exercícios

Criar três tabelas no banco de dados: **funcionarios**, **pagamentos** e **descontos**.

funcionarios

1	Luis
2	Marina
3	Letícia
4	Gustavo
5	Mateus

pagamentos

1	1	100
2	1	200
3	3	300
4	5	400
5	5	500

descontos

1	1	50
2	2	20
3	5	30

```
CREATE TABLE funcionarios(
  codigo_funcionario INT,
  nome                VARCHAR(50)
);
```

funcionarios

1	Luis
2	Marina
3	Letícia
4	Gustavo
5	Mateus

```
CREATE TABLE pagamentos(
  codigo_pagto      INT,
  codigo_funcionario INT,
  valor             DECIMAL(10,2)
);
```

pagamentos

1	1	100
2	1	200
3	3	300
4	5	400
5	5	500

```
CREATE TABLE descontos(
  codigo_desconto   INT,
  codigo_funcionario INT,
  valor             DECIMAL(10,2)
);
```

descontos

1	1	50
2	2	20
3	5	30

```
INSERT INTO funcionarios VALUES(1, "Luis");  
INSERT INTO funcionarios VALUES(2, "Marina");  
INSERT INTO funcionarios VALUES(3, "Letícia");  
INSERT INTO funcionarios VALUES(4, "Gustavo");  
INSERT INTO funcionarios VALUES(5, "Mateus");
```

```
INSERT INTO pagamentos VALUES(1, 1, 100);  
INSERT INTO pagamentos VALUES(2, 1, 200);  
INSERT INTO pagamentos VALUES(3, 3, 300);  
INSERT INTO pagamentos VALUES(4, 5, 400);  
INSERT INTO pagamentos VALUES(5, 5, 500);
```

```
INSERT INTO descontos VALUES(1, 1, 50);  
INSERT INTO descontos VALUES(2, 2, 20);  
INSERT INTO descontos VALUES(3, 5, 30);
```

Exemplo de INNER JOIN

```
SELECT f.nome AS Funcionário,  
       p.valor AS Pagamento  
FROM funcionarios f  
INNER JOIN pagamentos p ON f.codigo_funcionario = p.codigo_funcionario;
```

nome	pagamento
Luis	100.00
Luis	200.00
Letícia	300.00
Mateus	400.00
Mateus	500.00
5 rows in set (0.00 sec)	

Apesar de termos cinco funcionários na tabela, ele mostrou apenas três, o motivo é que apenas estes três tem pagamentos. Veja que o INNER JOIN fez uma junção entre **funcionarios** e **pagamentos** e desconsiderou os funcionários sem pagamentos.

INNER JOIN com três tabelas

```
SELECT f.nome AS Funcionário,  
       p.valor AS Pagamento,  
       d.valor AS Desconto  
FROM funcionarios f  
INNER JOIN pagamentos p ON f.codigo_funcionario = p.codigo_funcionario  
INNER JOIN descontos d ON f.codigo_funcionario = d.codigo_funcionario;
```

nome	pagamento	desconto
Luis	100.00	50.00
Luis	200.00	50.00
Mateus	400.00	30.00
Mateus	500.00	30.00

4 rows in set (0.00 sec)

Neste caso apenas dois funcionários foram mostrados já que incluímos na consulta os descontos, ou seja, a leitura que esta consulta fez é: mostrar funcionários que tem pagamentos e descontos.

Exemplo de LEFT JOIN

```
SELECT f.nome AS Funcionário,  
       p.valor AS Pagamento  
FROM funcionarios f  
LEFT JOIN pagamentos p ON f.codigo_funcionario = p.codigo_funcionario;
```

nome	pagamento
Luis	100.00
Luis	200.00
Letícia	300.00
Mateus	400.00
Mateus	500.00
Marina	NULL
Gustavo	NULL

7 rows in set (0.00 sec)

Os funcionários 3 e 5 não tem pagamentos, mas ainda assim eles apareceram na consulta, já que a função LEFT JOIN considera apenas a coluna da esquerda e retorna NULL (nulo) quando a coluna da direita não tiver um valor correspondente.

Incluindo o desconto...

```
SELECT f.nome AS Funcionário,  
       p.valor AS Pagamento,  
       d.valor AS Desconto  
FROM funcionarios f  
LEFT JOIN pagamentos p ON f.codigo_funcionario = p.codigo_funcionario  
LEFT JOIN descontos d ON f.codigo_funcionario = d.codigo_funcionario;
```

nome	pagamento	desconto
Luis	100.00	50.00
Luis	200.00	50.00
Mateus	400.00	30.00
Mateus	500.00	30.00
Letícia	300.00	NULL
Marina	NULL	20.00
Gustavo	NULL	NULL

7 rows in set (0.00 sec)

O que fizemos foi uma espécie de LEFT JOIN em cascata.

Útil quando queremos partir de uma base (**funcionarios**) e listar todas as correspondências ou não das tabelas (**pagamentos** e **descontos**) a ela relacionadas

Dicas

Fazendo “perguntas” ao MySQL

```
SELECT VERSION();      -> 8.3.0
SELECT CURRENT_DATE;   -> 2024-10-28
SELECT USER();         -> root@localhost
SELECT NOW();          -> 2024-10-28 10:15:42
```

Podemos perguntar 2 coisas ao mesmo tempo:

```
SELECT VERSION(), CURRENT_DATE;
```

```
+-----+-----+
| VERSION() | CURRENT_DATE |
+-----+-----+
| 8.3.0      | 2024-10-28   |
+-----+-----+
```

Extras

Podemos usar MySQL como calculadora. Digite na linha de comando:

```
SELECT 2+2;
```

-> 4

```
SELECT PI();
```

-> 3.141593

```
SELECT COS(PI());
```

-> -1

```
SELECT COS(PI()/2);
```

-> 6.123233995736766e-17

```
SELECT COS(PI())/2;
```

-> -0.5

Extras

```
SELECT ROUND (PI () ,2) ; -- Definindo o número de casa decimais
```

-> 3.14

```
SELECT ROUND (PI () ,10) ;
```

-> 3.1415926536

```
SELECT SQRT (9) ;
```

-> 3

```
SELECT ABS (-2) ;
```

-> 2

```
SELECT MOD (5 ,2) ;
```

-> 1

```
SELECT POWER (3 ,2) ;
```

-> 9

Funções String

```
SELECT CONCAT ( 'My' , 'S' , 'QL' );  
      -> MySQL
```

```
SELECT CONCAT_WS ( '-', 'Uni SENAC' , 'Campus' , 'Pelotas' );  
      -> Uni SENAC-Campus-Pelotas
```

```
SELECT REPEAT ( 'MySQL' , 3 );  
      -> MySQLMySQLMySQL
```

```
SELECT CHAR_LENGTH ( 'Uni SENAC' );  
      -> 9
```

MySQL – Função SUBSTRING()

Extrai um número específico de caracteres a partir de uma posição

Sintaxe: **SUBSTRING**(string, posição, tamanho)

posição left (+) →	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
string	w	w	w	.	s	e	n	a	c	r	s	.	c	o	m	.	b	r	tamanho=18
	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	← posição right (-)

MySQL – Função SUBSTRING()

posição left (+) →	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
string	w	w	w	.	s	e	n	a	c	r	s	.	c	o	m	.	b	r	tamanho=18
	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	← posição right (-)

Exemplo01: **SUBSTRING**("www.senacrs.com.br", 5, 7);

Resultado: **senacrs**

Exemplo02: **SUBSTRING**("www.senacrs.com.br", 5);

Resultado: **senacrs.com.br**

MySQL – Função SUBSTRING()

posição left (+) →	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
string	w	w	w	.	s	e	n	a	c	r	s	.	c	o	m	.	b	r	tamanho=18
	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	← posição right (-)

Exemplo03: **SUBSTRING**("www.senacrs.com.br", -5);

Resultado: **om.br**

Exemplo04: **SUBSTRING**("www.senacrs.com.br", -10, 6);

Resultado: **crs.co**

MySQL – Função SUBSTRING()

posição left (+) →	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
string	w	w	w	.	s	e	n	a	c	r	s	.	c	o	m	.	b	r	tamanho=18
	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	← posição right (-)

Exemplo05: **SUBSTRING**("www.senacrs.com.br" FROM 5 FOR 7);
Resultado: **senacrs**

Exemplo01: **SUBSTRING**("www.senacrs.com.br", 5, 7);
Resultado: **senacrs**

Funções String

```
SUBSTRING_INDEX(str, delim, count)
SELECT SUBSTRING_INDEX('www.mysql.com', '.', 2);
      -> 'www.mysql'
```

```
SELECT SUBSTRING_INDEX('www.mysql.com', '.', -2);
      -> 'mysql.com'
```

Retorna a *substring* da *string* 'www.mysql.com' por exemplo antes de 2 ocorrências do delimitador .

Se `count` é positivo, tudo a esquerda do delimitador final (contando a partir da esquerda) é retornado.

Se `count` é negativo, tudo a direita do delimitador final (contando a partir da direita) é retornado.

Funções String

```
SELECT REVERSE ( ' abc ' ) ;  
      -> ' cba '
```

```
SELECT UCASE ( ' Paulo ' ) ;  
      -> ' PAULO '
```

```
SELECT UPPER ( ' Paulo ' ) ;  
      -> ' PAULO '
```

```
SELECT LCASE ( ' MYSQL ' ) ;  
      -> ' mysql '
```

```
SELECT LOWER ( ' MYSQL ' ) ;  
      -> ' mysql '
```

Funções Data

```
SELECT DAYOFWEEK ( '2024-10-28' ) ;
```

→ **2** OBS: (1 = Domingo, 2 = Segunda, ... 7 = Sábado)

```
SELECT WEEKDAY ( '2024-10-28' ) ;
```

→ **0** OBS: (0 = Segunda, 1 = Terça, ... 6 = Domingo)

```
SELECT DAYOFMONTH ( '2024-10-28' ) ;
```

→ **28**

```
SELECT MONTH ( '2024-10-28' ) ;
```

→ **10**

Funções Data

```
SELECT DAYNAME ( ' 2024-10-28 ' ) ;  
      -> 'Monday'
```

```
SELECT MONTHNAME ( ' 2024-10-28 ' ) ;  
      -> 'October'
```

+ Exercícios

Exercícios – Parte 1

1 – Escreva um script SQL para criar uma tabela chamada **empregado** (de acordo com a estrutura apresentada):

Campo	Tipo	Descrição
codigo	Integer	Código do funcionário(não nulo)
nome	Char(40)	Nome do funcionário (não nulo)
setor	Char(2)	Setor onde o funcionário trabalha
cargo	Char(20)	cargo do funcionário
salario	Decimal(10,2)	salário do funcionário
Chave Primária		Será o campo codigo

Exercícios – Parte 2

1 – Crie instruções SQL para inserir os registros (da imagem) na tabela criada

codigo	nome	setor	cargo	salario
1	Cleide Campos	1	Secretária	1000.00
3	Andreia Batista	6	Programadora	1500.00
4	Cristiano Souza	6	Programador	1500.00
6	Mario Souza	4	Analista	2200.00
7	Ana Silva	4	Secretária	1000.00
9	Silvia Moraes	5	Supervisora	1650.00
10	José da Silva	1	Programador	1500.00
15	Manoel Batista	1	Projetista	2500.00
25	João Silva	4	Supervisor	1650.00

9 rows in set (0.00 sec)

Exemplo:

```
INSERT INTO empregado VALUES (1, 'Cleide Campos', '1', 'Secretária',1000 );
```


Exercícios – Parte 3

Crie as instruções SQL para:

1. Apresentar a listagem completa dos registros da tabela **empregado**;
2. Apresentar uma listagem dos **nomes** e dos **cargos** de todos os registros da tabela **empregado**;
3. Apresentar uma listagem dos **nomes** dos empregados do **setor 1**
4. Listagem dos **nomes** e dos **salários** por **ordem** de **nome** (**A-Z**)
5. Listagem dos **nomes** e dos **salários** por **ordem** de **nome** em formato descendente (**Z-A**)
6. Listagem dos **setores** e **nomes** colocados por **ordem** do campo **setor** em formato **ascendente** e do campo **nome** em formato **descendente**.
7. Listagem de **nomes ordenados** pelo campo **nome** em formato **ascendente**, dos empregados do **setor 4**.

Exercícios – Parte 4

Crie as instruções SQL para:

1. O empregado de **código 7** teve um **aumento** de **salário** para **2500,50**.
2. **Andreia Batista** foi **transferida** do departamento 5 para o **departamento 3**.
3. **Todos os empregados** da empresa tiveram um **aumento** de salário de **20%**.
4. **Todos os empregados** do **setor 1** foram demitidos , **exclua-os**.
5. **Mario Souza** pediu demissão, **exclua-o**.

Exercícios – Parte 5

Crie as instruções SQL para:

1. Apresentar **nome** e **salário** dos empregados que **ganham acima** de **1700.00**.
2. Listar os **empregados** do **setor 5**.
3. Listar os **empregados** cujo **cargo** é **programador**.
4. Listar **empregados** com **salário até 2000,00**.

Exercícios – Parte 6

Crie as instruções SQL para:

1. Listar **programadores** do **setor 2**.
2. Listar empregados que sejam **supervisor** ou **supervisora**.
3. Listar empregados que **não** sejam **gerentes**.

Exercícios – Parte 7

Crie as instruções SQL para:

1. Listar empregados cujo **nome comece** com a letra **A**
2. Listar empregados cujo **nome** tem a **segunda** letra **A**
3. Listar empregados que tem a sequência **AN** em **qualquer posição** do **nome**.

Exercícios – Parte 8

Crie as instruções SQL para exibir a(o):

1. **Média** aritmética dos **salários** de todos os empregados
2. **Média** aritmética dos **salários** de todos os empregados do **setor 3**
3. **Soma** dos **salários** de todos os empregados
4. **Soma** dos **salários** de todos os empregados do **setor 5**
5. **Maior salário** existente entre todos os empregados
6. **Menor salário** existente entre todos os empregados
7. **Número** de **empregados** do **setor 3**
8. **Número** de **empregados** que ganham **mais que 2000,00**
9. **Número** de **setores** existentes no cadastro de empregados.

CINEMINHA

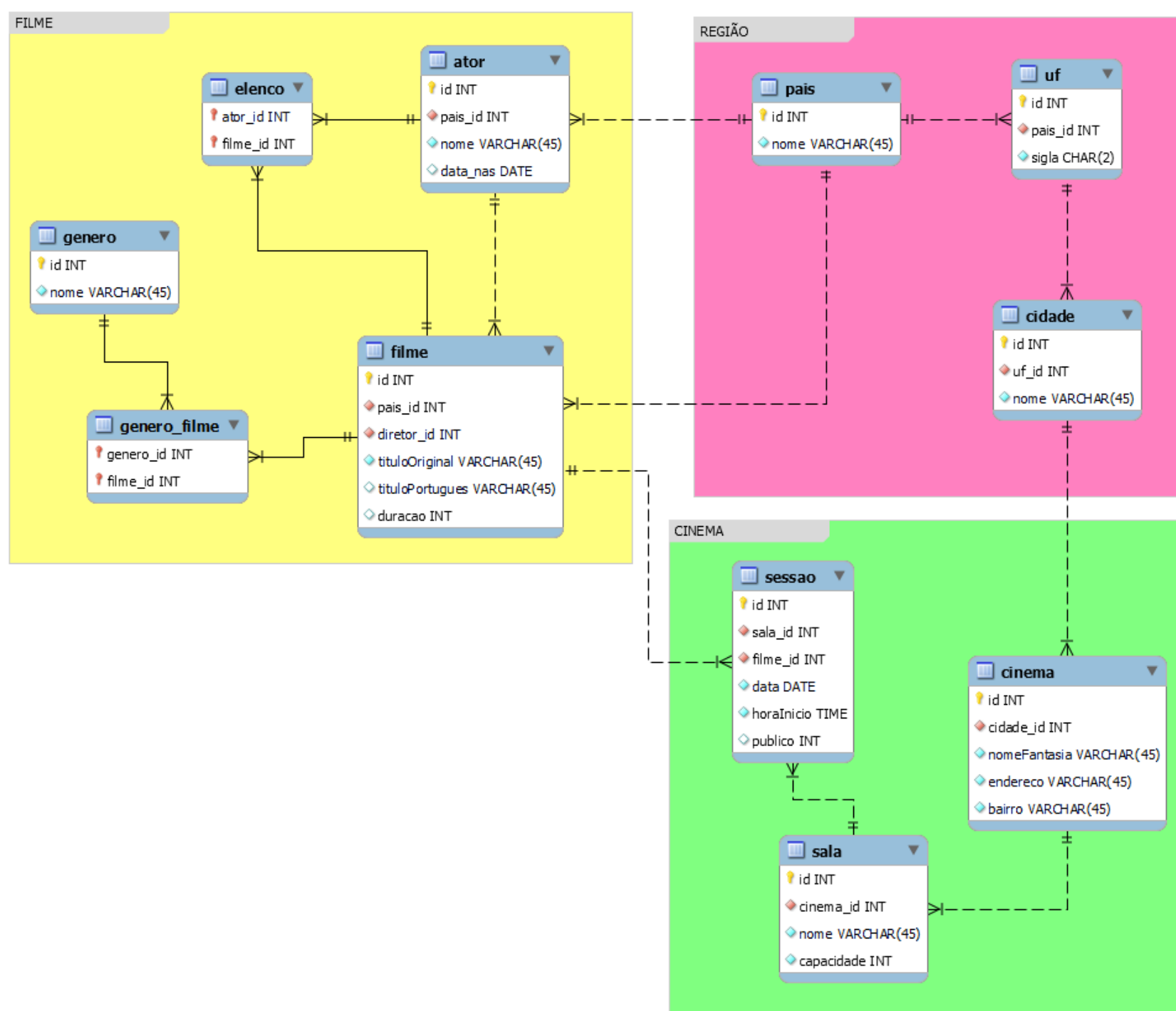
CINEMINHA

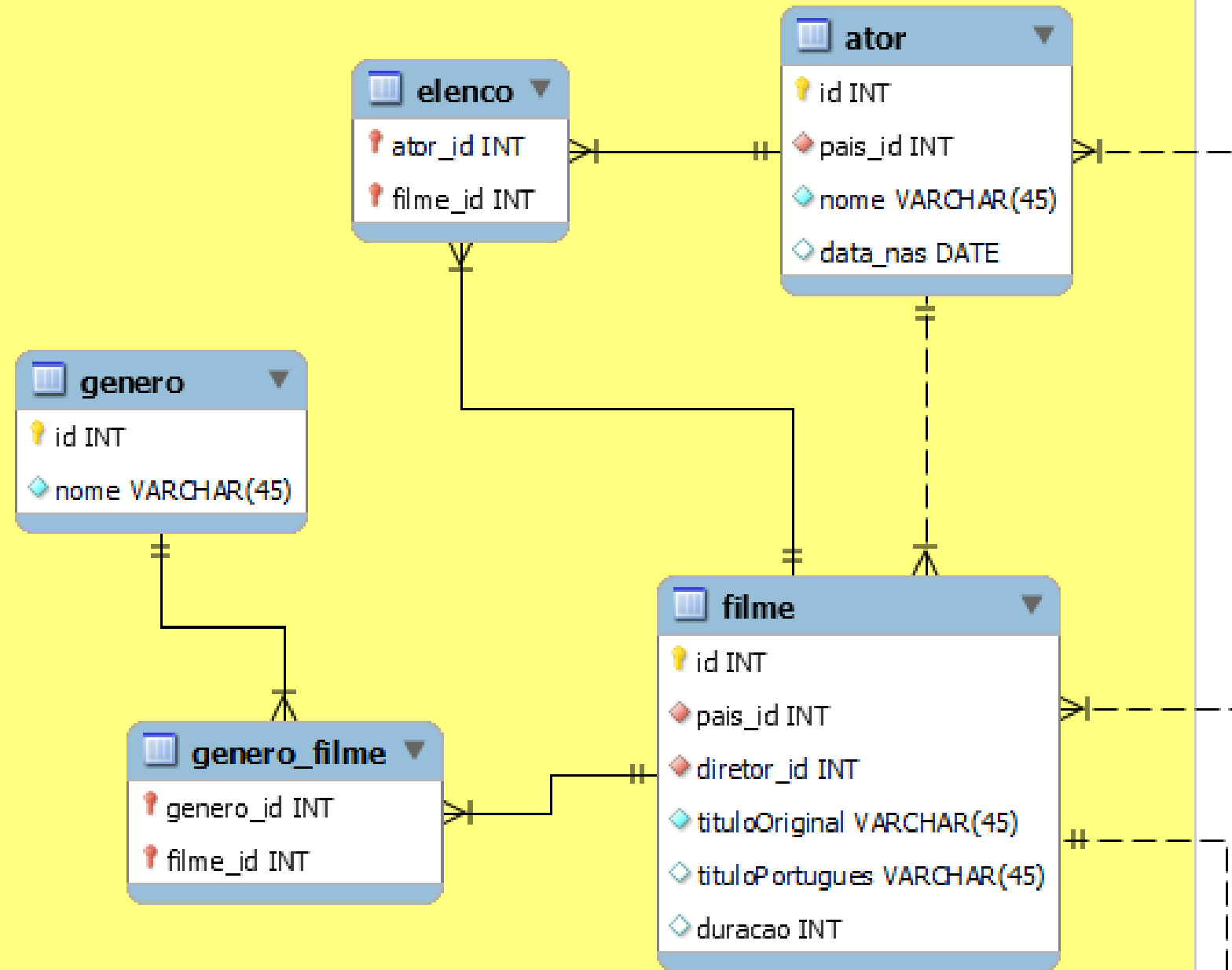
O Banco de dados a seguir foi projetado para se adaptar às necessidades de um controle de cinemas e filmes da empresa CINEMINHA.

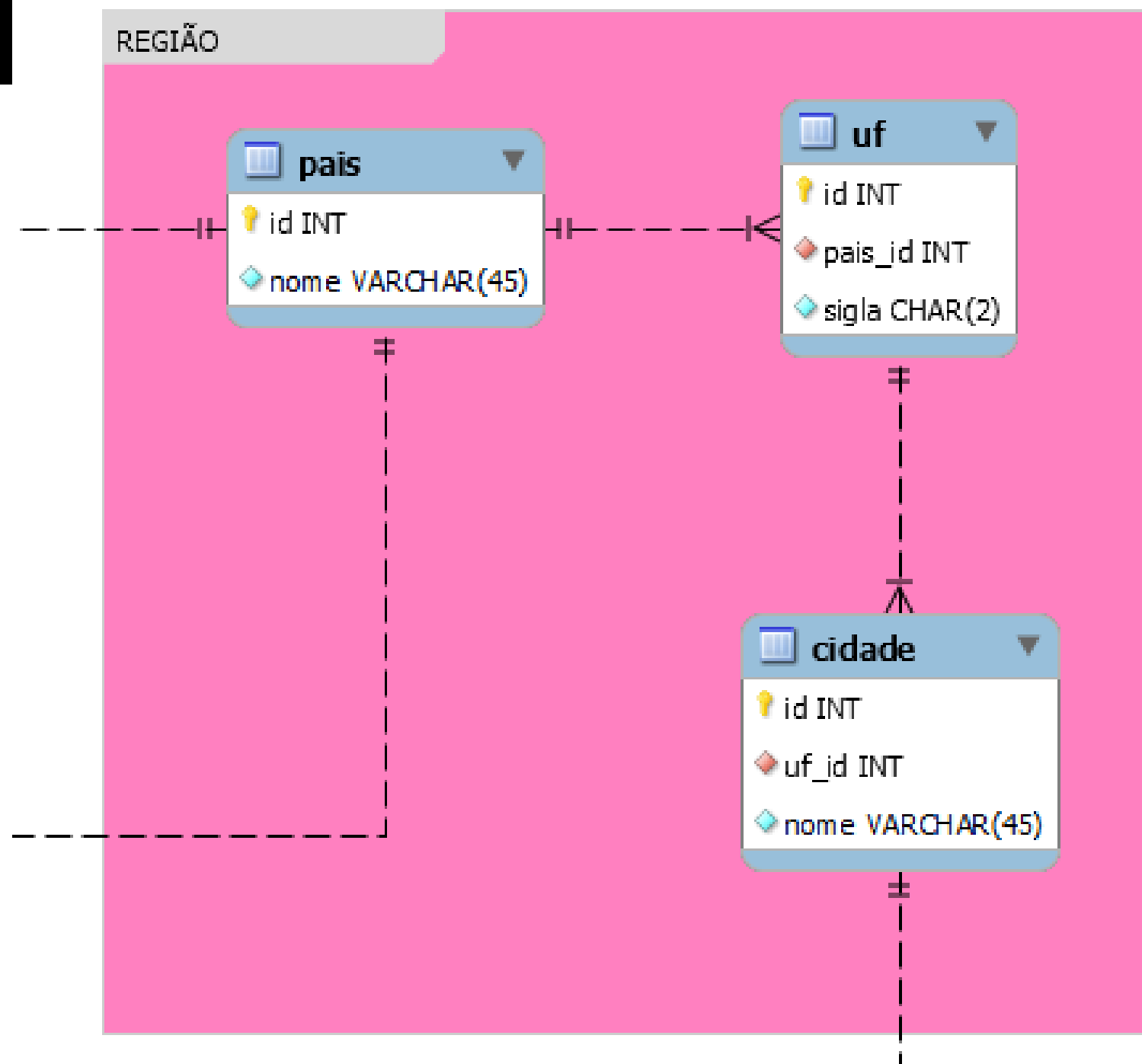
A modelagem, seguiu algumas definições e regras:

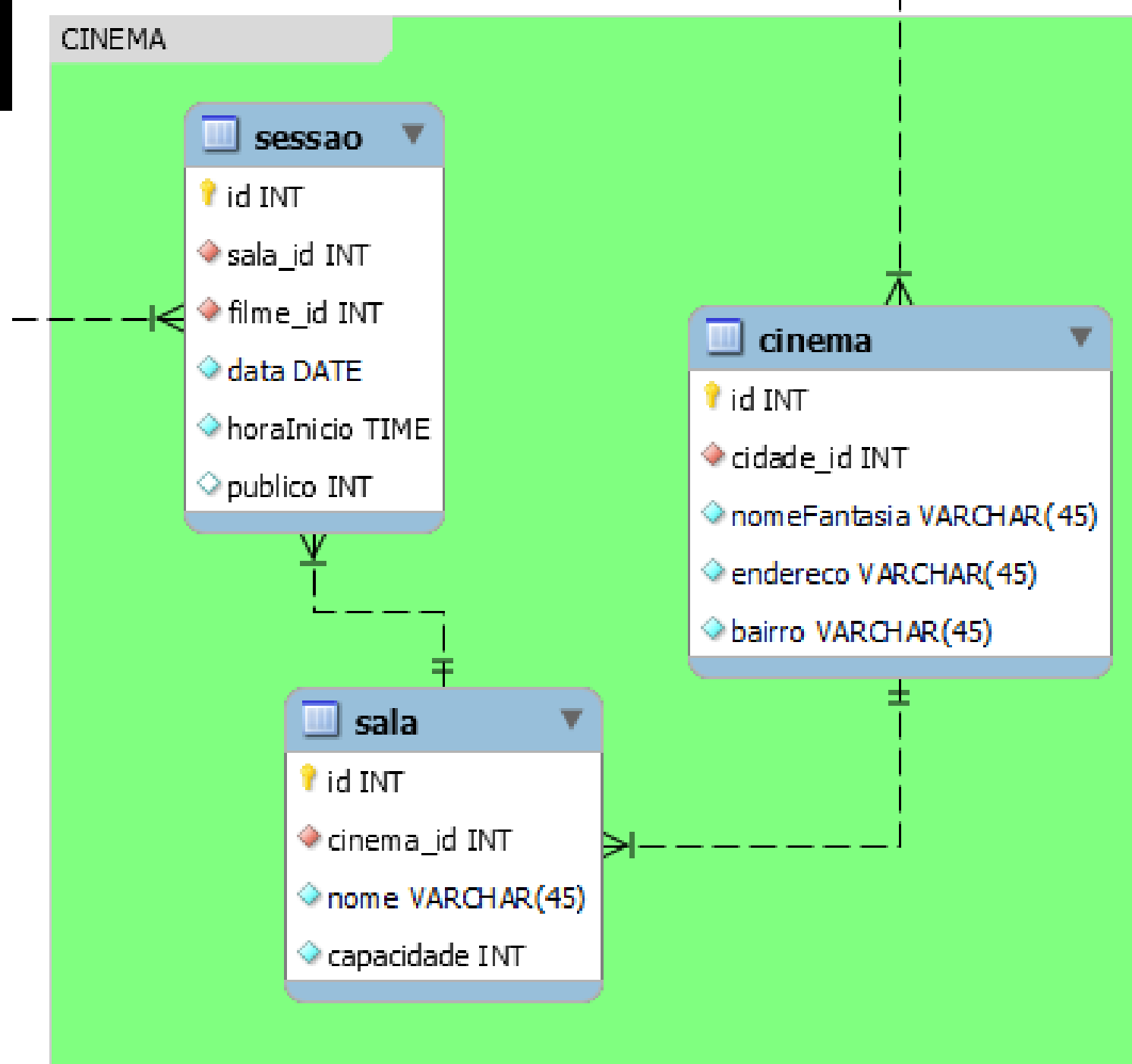
1. A empresa de distribuição possui vários cinemas em várias localidades;
2. Cada cinema possui uma identificação única, um nome fantasia e um endereço;
3. Cada cinema possui “n” salas. Cada uma com identificação única e sua capacidade de lotação;
4. Os filmes podem ser dos mais variados tipos e gêneros;
5. Cada filme é registrado com um título original, e se for filme estrangeiro, possuirá também o título em português, o gênero, sua duração, seu país de origem, informações sobre os atores que compõem seu elenco, e o seu diretor. Existe um único diretor para cada filme;
6. Alguns cinemas apresentam mais de um filme em cartaz (em sessões e/ou salas diferentes);
7. As sessões possuem horários determinados;
8. Os atores de um filme podem, obviamente, atuar em diversos filmes, assim como o diretor de um filme pode também ser ator nesse filme, ou ator em outro filme. Um ator possui como característica de identificação: nome, nacionalidade e data de nascimento;
9. As sessões do cinema devem ter seu público registrado diariamente, para que se permita a totalização do público quando o filme sair de cartaz, ou a qualquer instante.

CINEMINHA









Crie as consultas

- a) Apuração do público por município
- b) Apuração do público por cinema
- c) Apuração do público por sessão de cada cinema;
- d) Permitir uma forma de consulta, que dado um determinado ator, sejam localizados todos os cinemas onde estão (ou estiveram) em cartaz os filmes em que este ator atua;
- e) Em quais cinemas está sendo exibido um determinado gênero de filme;
- f) Em quais cinemas estão sendo exibidos filmes nacionais;